Natural Computing (2023) 22:645–657
https://doi.org/10.1007/s11047-023-09953-2

Fast elitist ABC for makespan optimisation in interval JSP

Hernán Díaz[1] · Juan José Palacios[1] · Inés González-Rodríguez[2] · Camino R. Vela[1]

Accepted: 11 July 2023 / Published online: 5 August 2023
© The Author(s) 2023

Abstract

This paper addresses a variant of the Job Shop Scheduling Problem with makespan minimisation where uncertainty in task durations is taken into account and modelled with intervals. A novel Artificial Bee Colony algorithm is proposed where the classical layout is simplified, increasing the algorithm's speed and reducing the number of parameters to set up. We also take into account the fundamental principles of exploration around a local solution and attraction to a global solution to improve diversity in the hive. The increase on speed and diversity allows to include a Local Search phase to better exploit promising areas of the search space. A parametric analysis is conducted and the contribution of the new strategies is analysed. The results of the new approach are competitive with those obtained with previous methods in the literature, but taking less runtime. The addition of Local Search improves the results even further, outperforming the best-known ones from the literature. An additional sensitivity study is conducted to assess the advantages of considering uncertainty and how increasing it affects the solution's robustness.

Keywords Job shop scheduling · Makespan · Interval uncertainty · Artificial Bee colony · Robustness

1 Introduction

The job shop scheduling problem (JSP) is considered to be one of the most relevant scheduling problems. It consists in allocating a set of resources to execute a set of jobs under a set of given constraints, with the most popular objective in the literature being the minimisation of the project's execution timespan, also known as makespan. Solving this problem improves the efficiency of chain production processes, optimising the use of energy and materials Pinedo (2016) and having a positive impact on costs and environmental sustainability. However, in real-world applications, the available information is often imprecise. Interval uncertainty arises as soon as information is incomplete, and contrary to the case of stochastic and fuzzy scheduling, it does not assume any further knowledge, thus representing a first step towards solving problems in other frameworks Allahverdi et al. (2014). Moreover, intervals are a natural model whenever decision-makers prefer to provide only a minimal and a maximal duration, and obtain interval results that can be easily understood. Under such circumstances, interval scheduling allows to concentrate on significant scheduling decisions and to produce robust solutions.

Contributions to interval scheduling in the literature are not abundant. In Lei (2012), a genetic algorithm is proposed for a JSP minimizing the total tardiness with respect to job due dates with both processing times and due dates represented by intervals. In Díaz et al. (2022), a different genetic algorithm is applied to the same problem, including a study of different interval ranking methods based on the robustness of the resulting schedules. A population-based neighbourhood search for an interval JSP with makespan minimisation is presented in Lei (2011). In Li et al. (2019), a hybrid between particle swarm and a genetic algorithm is used to solve a flexible JSP with interval processing times

✉ Juan José Palacios
palaciosjuan@uniovi.es

Hernán Díaz
diazhernan@uniovi.es

Inés González-Rodríguez
gonzalezri@unican.es

Camino R. Vela
crvela@uniovi.es

[1] Department of Computing, University of Oviedo, Asturias, Spain

[2] Department of Matemáticas, Estadística y Computación, Universidad de Cantabria, Cantabria, Spain

as part of a larger integrated planning and scheduling problem. More recently, a genetic algorithm is applied in Díaz et al. (2020) to the JSP with interval uncertainty minimizing the makespan and two different algorithms based on artificial bee colonies are proposed in Díaz et al. (2022) and Díaz et al. (2023) for the same problem.

Due to the complexity of job shop scheduling problems, metaheuristic search methods are especially suitable to solve them. In particular, Artificial Bee Colony (ABC) is a swarm intelligence optimiser inspired by the intelligent foraging behaviour of honeybees that has shown very competitive performance on JSP with makespan minimisation. For instance, Wong et al. Wong et al. (2008) propose an evolutionary computation algorithm based on ABC that includes a state transition rule to construct the schedules. Taking some principles from Genetic Algorithms, Yao et al. Yao et al. (2010) present an Improved ABC (IABC) where a mutation operation is used for exploring the search space, enhancing the search performance of the algorithm. Later, Banharnsakun et al. Banharnsakun et al. (2012) propose an effective ABC approach based on updating the population using the information of the best-so-far food source. In Díaz et al. (2022), an elitism mechanism is introduced to increase diversity and solve an interval job shop problem with makespan minimisation. The same problem is tackled in Díaz et al. (2023) introducing the seasonal behaviour of honeybees as part of the onlooker bee phase.

In the following, we extend the work presented in Díaz et al. (2022) to solve the interval JSP with makespan minimisation. We propose several improvements on the introduced Elite ABC method that aim at speeding up the algorithm while increasing its diversity. A Local Search is then included into the ABC to exploit the new diversity and obtain better results. The robustness study conducted in Díaz et al. (2022) is complemented with a new sensitivity analysis to assess the quality of solutions in scenarios of increasing uncertainty. The rest of the paper is organized as follows: the interval JSP is presented in Sect. 2; in Sect. 3 we describe the different components that conform the ABC algorithm that addresses this problem; in Sect. 4 we compare these strategies and the best one is also compared with the state of the art; a sensitivity analysis is also included in Sect. 4.

2 The job shop problem with interval durations

The classical *job shop scheduling problem* consists of a set of resources $M = \{M_1, \ldots, M_m\}$ and a set of jobs $J = \{J_1, \ldots, J_n\}$. Each job J_j is organised in tasks or operations $(o(j,1), \ldots, o(j, m_j))$ that need to be sequentially scheduled. We assume w.l.o.g. that tasks are indexed from 1 to $N = \sum_{j=1}^{n} m_j$, so we can refer to task $o(j, l)$ by its index $o = \sum_{i=1}^{j-1} m_i + l$ and denote the set of all tasks as $O = \{1, \ldots, N\}$. Each task $o \in O$ requires the uninterrupted and exclusive use of a machine $v_o \in M$ for its whole processing time p_o.

A solution to this problem is a *schedule* $\boldsymbol{s}$, i.e. an allocation of starting times for each task, which, besides being *feasible* (all constraints hold), is *optimal* according to some criterion, in our case, minimal makespan C_{max}.

2.1 Interval uncertainty

Following Lei (2011) and Díaz et al. (2020), uncertainty in the processing time of tasks is modelled using closed intervals. Therefore, the processing time of task $o \in O$ is represented by an interval $\mathbf{p_o} = [\underline{p}_o, \overline{p}_o]$, where $\underline{p}_o$ and $\overline{p}_o$ are the available lower and upper bounds for the exact but unknown processing time p_o.

The interval JSP (IJSP) with makespan mimisation requires two arithmetic operations: addition and maximum. Given two intervals $\mathbf{a} = [\underline{a}, \overline{a}], \mathbf{b} = [\underline{b}, \overline{b}]$, the addition is expressed as $[\underline{a} + \underline{b}, \overline{a} + \overline{b}]$ and the maximum as $[\max(\underline{a}, \underline{b}), \max(\overline{a}, \overline{b})]$. Also, given the lack of a natural order in the set of closed intervals, to determine the schedule with the "minimal" makespan, we need an interval ranking method. For the sake of fair comparisons with the literature, we shall use the midpoint method: $\mathbf{a} \leq_{MP} \mathbf{b} \Leftrightarrow m(\mathbf{a}) \leq m(\mathbf{b})$ with $m(\mathbf{a}) = (\underline{a} + \overline{a})/2$. This is used in Díaz et al. (2020, 2022, 2023) and it is equivalent to the ranking method used in Lei (2012) and Lei (2011). Notice that $m(\mathbf{a})$ coincides with the expected value of the uniform distribution on the interval, $E[\mathbf{a}]$.

A schedule $\boldsymbol{s}$ for the IJSP establishes a relative order π among tasks requiring the same machine. Conversely, given a task processing order π the schedule $\boldsymbol{s}$ may be computed as follows. For every task $o \in O$, let $\mathbf{s_o}(\pi)$ and $\mathbf{c_o}(\pi)$ denote respectively the starting and completion times of o, let $PM_o(\pi)$ and $SM_o(\pi)$ denote the predecessor and successor tasks of o in the machine v_o according to π, and let PJ_o and SJ_o denote the tasks preceding and succeeding o in its job. Then the starting time of o is given by $\mathbf{s_o}(\pi) = \max(\mathbf{s}_{PJ_o} + \mathbf{p}_{PJ_o}, \mathbf{s}_{PM_o(\pi)} + \mathbf{p}_{PM_o(\pi)})$, and the completion time by $\mathbf{c_o}(\pi) = \mathbf{s_o}(\pi) + \mathbf{p_o}$. The makespan is computed as the completion time of the last task to be processed according to π thus, $\mathbf{C_{max}}(\pi) = \max_{o \in O}\{c_o(\pi)\}$. If there is no possible confusion regarding the processing order, we may simplify notation by writing $\mathbf{s_o}$, $\mathbf{c_o}$ and $\mathbf{C_{max}}$.

To illustrate this, we consider a problem with $n = 3$ jobs and $m_j = m = 2, j \in \{1, 2, 3\}$ machines. The following

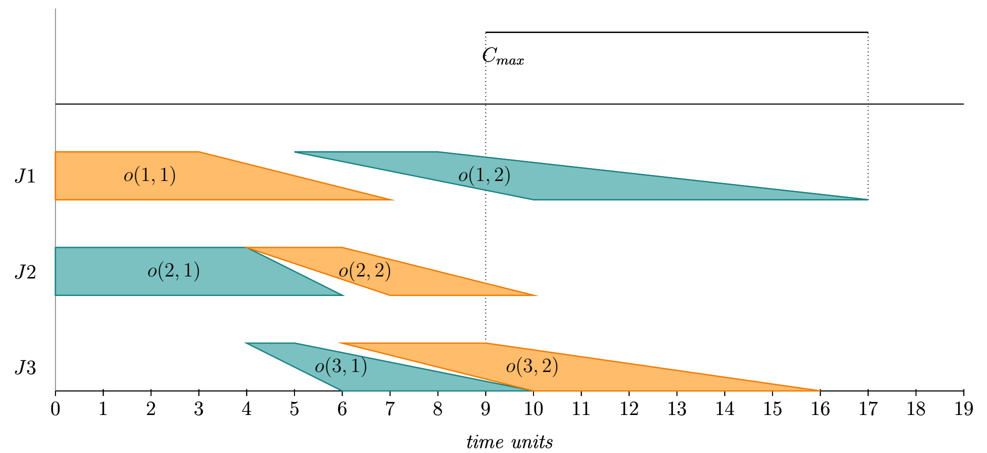


Fig. 1 Gantt Chart representing a solution to an IJSP instance

matrices show for each operation $o(j, l)$ its processing time $(p(j, l))$ and its required machine $(v(j, l))$:

$$p = \begin{pmatrix} [3,7] & [3,7] \\ [4,6] & [2,3] \\ [1,4] & [3,6] \end{pmatrix} \qquad v = \begin{pmatrix} M_1 & M_2 \\ M_2 & M_1 \\ M_2 & M_1 \end{pmatrix}$$

Following the notation introduced above, the second operation of job J_3, $o(3, 2)$, can be referred to as $o = 6$, so $\mathbf{p_6} = p(2,3) = [3,6]$ and $v_6 = v(2,3) = M_1$. Given the task processing order $\pi = (1,3,5,4,2,6)$, operation 1 ($o(1, 1)$) is scheduled first. Since it has no job or machine predecessor, it is scheduled at instant $\mathbf{s_1} = [0,0]$ and $\mathbf{c_1} = \mathbf{s_1} + \mathbf{p_1} = [3,7]$. The same occurs with operation 3 ($o(2, 1)$): $\mathbf{s_3} = [0,0]$ and $\mathbf{c_3} = [4,6]$. Operation 5 ($o(3, 1)$) requires the same machine as the already scheduled operation 3 (M_2), so $PM_5 = 3$. Since $o = 5$ has no job predecessor, $\mathbf{s_5} = \mathbf{s_3} + \mathbf{p_3} = [0,0] + [4,6] = [4,6]$ and $\mathbf{c_5} = [5,10]$. Operation 4, with $PJ_4 = 3$, requires the same machine as operation 1, so $PM_4 = 1$ and $\mathbf{s_4} = \max(\mathbf{s_3} + \mathbf{p_3}, \mathbf{s_1} + \mathbf{p_1}) = \max([4,6],[3,7]) = [4,7]$ ($\mathbf{c_4} = [6,10]$). Scheduling the remaining operations following π, we obtain that $\mathbf{s_2} = [5,10]$, $\mathbf{c_2} = [8,17]$, $\mathbf{s_6} = [6,10]$, $\mathbf{c_6} = [9,16]$. Consequently, the makespan is $\mathbf{C_{max}} = \max_{o \in O}\{\mathbf{c_o}\} = [9,17]$ and its midpoint, $m(\mathbf{C_{max}}) = 13$

A Gannt chart of the schedule that results from the processing order π can be seen in Fig. 1. It adapts the chart to the interval framework, in the same manner as it is proposed to do in Fortemps and Roubens (1996) for fuzzy processing times.

2.2 Robustness on interval JSP

In a solution to the IJSP, the makespan value is not an exact value, but an interval. It is only after the solution is executed on a real scenario that actual processing times for tasks $P^{ex} = \{p_o^{ex} \in [\underline{p}_o, \overline{p}_o], o \in O\}$ are known. Therefore, it is not until that moment that the actual makespan $C_{max}^{ex} \in [\underline{C}_{max}, \overline{C}_{max}]$ can be found. It is desirable that this executed makespan C_{max}^{ex} does not differ much from the expected value of the makespan according to the interval $\mathbf{C_{max}}$.

This is the idea behind the concept of ϵ-robustness first proposed in Bidot et al. (2009) for stochastic scheduling, and later adapted to the IJSP in Díaz et al. (2020). For a given $\epsilon \geq 0$, a schedule with makespan $\mathbf{C_{max}}$ is considered to be *ϵ-robust* in a real scenario P^{ex} if the relative error made by the expected makespan $E[\mathbf{C_{max}}]$ with respect to the makespan C_{max}^{ex} of the executed schedule is bounded by ϵ, that is:

$$\frac{|C_{max}^{ex} - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]} \leq \epsilon. \tag{1}$$

Clearly, the smaller the bound ϵ, the more robust the interval schedule is. This measure of robustness is dependent on a specific configuration P^{ex} of task processing times obtained upon execution of the predictive schedule $\boldsymbol{s}$. In the absence of real data, as is the case with the usual synthetic benchmark instances for job shop, we may resort to Monte-Carlo simulations. We simmulate K possible configurations $P^k = \{p_o^k \in [\underline{p}_o, \overline{p}_o], o \in O\}$ using uniform probability distributions to sample durations for every task and compute for each configuration $k = 1, \ldots, K$ the exact makespan C_{max}^k that results from executing tasks according to the ordering provided by $\boldsymbol{s}$. Then, the average ϵ-robustness of the predictive schedule across the K possible configurations, denoted $\overline{\epsilon}$, can be calculated as:

$$\overline{\epsilon} = \frac{1}{K}\sum_{k=1}^{K} \frac{|C_{max}^k - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]}. \tag{2}$$

This value provides an estimate of how robust the solution $\boldsymbol{s}$ is across different processing times configurations.

3 Fast elitist artificial Bee colony

The Artificial Bee Colony Algorithm is a bioinspired swarm metaheuristic for optimisation based on the foraging behaviour of honey bees. Since it was introduced in Karaboga (2005) it has been successfully adapted to a variety of problems Karaboga et al. (2014).

Typically, the ABC starts by generating and evaluating an initial hive H_0 of random food sources. The best food source *Best* is assigned to the hive's queen. Then, the algorithm iterates over a number of cycles, each consisting of three phases mimicking the behaviour of three types of foraging bees: employed, onlooker and scout. In the employed bee phase, each food source is assigned to one employed bee, who explores a new candidate food source between its own food source and the queen's one. In the onlooker bee phase, each bee chooses a food source and tries to find a better one in its neighbourhood. At the end of each phase, the newly-found food source is evaluated. If it is equivalent to the queen's one (i.e. the best food source found so far), it is discarded for the sake of maintaining diversity in the hive. Otherwise, if it is better than the food source of the bee that generated it, it replaces it. If it cannot improve the original food source, then its *fs.numTrials* counter is increased by one. In the scout bee phase, if the number of improvement trials of a food source *fs.numTrials* reaches a given threshold NT_{max}, the scout bee determines a new food source to replace the former one in the hive of solutions. Typically, this is done by replacing the exhausted food source by a randomly generated one. The algorithm terminates when a certain stopping condition is met. In Díaz et al. (2022), this condition is met after a number *maxIter* of consecutive iterations without finding a food source that improves the queen's one.

In this general schema, diversity is mainly controlled by two mechanisms: modifying a random part of a food source to obtain a trail solution in the onlooker bee phase, or replacing a whole solution by a new one during the scout bee phase. Although it can be argued that these mechanisms help to avoid premature convergence, practical experiments have determined that this might not be the case for the IJSP Díaz et al. (2022). The employed and onlooker bee phases generate new solutions at each iteration, but they are included in the hive only if they can improve the food source from which they were generated. This may lead to a high selective pressure and facilitate getting trapped in local optima. When that happens, injecting a randomly generated solution in the scout bee phase with poor quality may not contribute enough to obtain better results. On the other hand, the current schema has up to three evaluation rounds, one at the end of each of the main phases. When diversity issues are present, most of these evaluations are useless, since new solutions won't be accepted, making the algorithm unnecessarily slow.

To increase diversity, an elitist selection mechanism was introduced in Díaz et al. (2022) so the employed bee phase does not always choose the queen's food source to explore, but a solution from a set of promising ones. In Díaz et al. (2023), an *ESABC* method is proposed where the onlooker bee phase is redefined based on the seasonal behaviour of honeybees to also increase the exploration capabilities of *ABC*. However, this seasonal behaviour is somehow similar to a Simulated Annealing method, so it increases the number of evaluations performed by the algorithm and therefore its overall complexity.

In Karaboga and Akay (2009), *ABC* is compared with other metaheuristics such as genetic algorithms (GA), differential evolution (DE) or particle swarm (PSO). In general, the exploration idea on these methods consists in altering all individuals of the population, or creating new ones, with a certain probability. Then the new set of solutions is evaluated and some type of replacement strategy is applied. This allows to first explore, and then apply the selective pressure through replacement. We propose to adapt this strategy to the setting of the *ABC* to increase diversity of solutions while keeping a reasonable complexity. The general layout of our proposal is inspired by the structure of Particle Swarm Optimization Kennedy and Eberhart (1995) in the sense that food sources can be understood as the local best position of a bee, and the queen's source would be the global best. At each iteration, bees explore new food sources (solutions) influenced both by the best sources of the hive (global best) and its current food source (local best). Thus, the employed bee phase and the onlooker bee phase are fused into one exploration step. At the end of the cycle, all new solutions are evaluated. Each bee moves to its new food source only if it is different from the queen's and better than its local best. If the new food source is not accepted, the bee increases its counter of trials *fs.numTrials* and when it exceeds the threshold NT_{max}, the bee moves to a random food source emulating scout bees. Allowing the bees to move more freely before evaluation can increase population's diversity and reduce the complexity of the *ABC*, going from three evaluation phases per iteration to only one. Furthermore, having only one replacement phase decreases the overall count of improvement trials and the number of random solutions introduced in the scouting section. We refer to this new algorithm as Fast Elitist Artificial Bee Colony (fEABC in short). To exploit the diversity and speed of the new structure, we propose to incorporate a new Local Search step before evaluation and do a further empirical evaluation on its advantages. The general structure of the algorithm is given in Algorithm 1. Each step is detailed in the following subsections.

Algorithm 1 Schema of the $fEABC$ Algorithm

Require: An IJSP instance
Ensure: A schedule
1: /*_Initial hive, see Section 3.1_*/
2: Generate and evaluate a hive H_0 of food sources
3: $Best \Leftarrow$ Best food source in H_0
4: $numIter \Leftarrow 0$
5: $i \Leftarrow 0$
6: **while** $numIter < maxIter$ **do**
7:   **for** each food source fs in H_i **do**
8:     /*_Exploration, see Section 3.2_*/
9:     $gBest \Leftarrow$ Select a solution from H_i using $Elite3$
10:     $new_{fs} \Leftarrow$ Find a neighbour of fs
11:     $new'_{fs} \Leftarrow$ Apply recombination to $(new_{fs}, gBest)$
12:     Evaluate new'_{fs}
13:     /*_Local search, see Section 3.3_*/
14:     $new'_{fs} \leftarrow$ Apply Local Search to new'_{fs}
15:     /*_Replacement, see Section 3.4_*/
16:     **if** new'_{fs} is better than $Best$ **then**
17:       $fs \Leftarrow new'_{fs}$
18:       $Best \Leftarrow new'_{fs}$
19:       $numIter \Leftarrow 0$
20:     **else**
21:       $numIter \Leftarrow numIter + 1$
22:       **if** new'_{fs} is better than fs and different than $Best$ **then**
23:         $fs \Leftarrow new'_{fs}$
24:       **else**
25:         $fs.numTrials \Leftarrow fs.numTrials + 1$
26:       **end if**
27:     **end if**
28:     **if** $fs.numTrials > \mathrm{NT}_{max}$ **then**
29:       $fs \Leftarrow$ Generate random solution
30:       Evaluate fs
31:       $fs.numTrials \Leftarrow 0$
32:     **end if**
33:   **end for**
34:   $i \leftarrow i + 1$
35: **end while**
36: **return** $Best$

3.1 Codification and initialization

We adopt the codification strategy from Díaz et al. (2020), where solutions are encoded using permutations with repetition Bierwirth (1995). Each solution s is represented by its task processing order π, but each operation $o(i, j)$ in π is replaced by its job number i. For example, for a problem with $n = 3$ jobs and $m = 2$ machines, a schedule with $\pi = (o(1,1), o(2,1), o(1,2), o(3,1), o(3,2), o(2,2))$ is encoded as (1, 2, 1, 3, 3, 2). To decode a solution, each value i in the permutation is replaced by the j-th task of that job, where j is the number of times the job has appeared so far in the permutation (e.g. the second time the value 1 appears, it refers to task $o(1, 2)$). To build a schedule from the permutation, we consider two decoding strategies. The strategy described in Sect. 2.1 can be seen as an adaptation to intervals of the concept of *Semi-active Schedule Generation Scheme*, or *Semi-active SGS*, introduced in Palacios et al. (2014) for the JSP with fuzzy durations. In this setting, the starting time $\mathbf{s}_o$ of each task o corresponds to the Earliest feasible Appending Starting time (ESA_o), and the resulting schedule is said to be *Semi-active* based on the definition from Sprecher et al. (1995). In an Insertion SGS, the starting time $\mathbf{s}_o$ of each task o in π is calculated as its Earliest feasible Insertion Starting time (ESI_o). Let $k = \nu_o$ be the machine where o needs to be processed, PJ_o the tasks preceding o in its job and $\sigma_k = (0, \sigma(1,k), \ldots, \sigma(\eta_k, k))$ the sequence of tasks already scheduled in machine ν_o. A feasible insertion position $q, 0 \leq q < \eta_k$ for o verifies that $max\{\overline{c}_{\sigma(q,k)}, \overline{c}_{PJ_o}\} +$

$\overline{p}_o \leq \overline{s}_{\sigma(q+1,k)}$ and $max\{\underline{c}_{\sigma(q,k)}, \underline{c}_{PJ_o}\} + \underline{p}_o \leq \underline{s}_{\sigma(q+1,k)}$. If such position exists, $ESI_o = max\{\mathbf{c}_{\sigma(q^*,k)}, \mathbf{c}_{PJ_o}\}$, where q^* is the smallest feasible insertion position. If there is no feasible insertion position, then $ESI_o = ESA_o$. The schedules that can be obtained with this decoding mechanism fall into the definition of *Active schedules* given in Sprecher et al. (1995); Palacios et al. (2014). The set of active schedules is smaller than the set of semi-active schedules and both are guaranteed to contain the optimal solution. This can be seen as an advantage, since reducing the search space makes it faster to navigate, but it can also decrease diversity in meta-heuristics working on that space. An empirical analysis is needed to find the best option.

To generate an initial hive H_0 for the algorithm, a set of food sources is created by randomly generating permutations with repetition that are feasible for the problem. These permutations are later decoded and evaluated using of the described *SGS*. When comparing the richness of two different food sources, the fitness function is used. Given two food sources fs and fs' encoding two schedules s and s' respectively, we consider that fs is better than fs' if $\mathbf{C_{max}}(s) \leq_{MP} \mathbf{C_{max}}(s')$.

3.2 Exploration strategy

At each iteration, each bee begins by exploring the neighbourhood around its currently-assigned food source fs. To generate new food sources in the surroundings of fs, a small change is performed using one of the following operators for permutations: Swap, Inversion or Insertion. Given the small magnitude of the changes, it is reasonable to expect that new solutions do not differ much from fs in terms of makespan but provide enough of a difference to increase diversity while maintaining the average quality of the population.

After moving to a neighbouring food source new_{fs}, the bee begins the exploration towards the best food sources known by the hive. In the classical *ABC*, the best food source is selected at the beginning of the iteration and is later used by all employed bees. In this work, each bee selects a food source *gBest* to move towards to. Selecting the best food source in the hive (the queen's) can lead to a shorter execution time derived from the lack of diversity in the solution bank Banharnsakun et al. (2012). Two alternatives were tested in our preliminary work in Díaz et al. (2022) to avoid this issue. *Elite2* consists on selecting the best food source among the group of sources with the highest number of improvement trials García-Álvarez et al. (2018). On the other hand, *Elite3* selects at random one of the best N food sources existing at the time, being N a configurable value. After an experimental study, the latter appears to be the most prominent strategy. Moreover, the fact of being able to configure the size of that set allows to balance exploration and exploitation. Therefore, we choose *Elite3* as selection strategy for each bee in our method.

Once the bee has selected its global best *gBest*, it applies a recombination operator to move from new_{fs} to *gBest*, obtaining a new source new'_{fs} containing information of both solutions. This focuses the exploration towards more promising areas of the search space. We test three different recombination operators especially tailored to Job Shop Scheduling Problems: Job-Order Crossover (JOX) Ono et al. (1996), Generalised Order Crossover (GOX) Bierwirth (1995) and Precedence Preservative Crossover (PPX) Bierwirth et al. (1996). Only after the bee has explored both its neighbouring food source and a solution towards the hive's bests, the newly food source $new_f s'$ is evaluated.

3.3 Local search

The diversity derived from the previous steps, and the reduction on the number of evaluations per iteration, creates an opportunity to include more exploitation-driven strategies such as Local Search. Local search techniques focus on exploitation to offer further improvements of solutions resulting from schedule generation heuristics. In our context, after the exploration phase of the bee is completed and a new solution new'_{fs} is evaluated, the bee may decide to carry an intense search in the vicinity of the new food source before moving on to the next iteration.

We take the neighbourhood structure defined in Van Laarhoven et al. (1992) as reference. There, a neighbour is generated by reversing a critical arc in the solution graph $G(\boldsymbol{s})$ representing schedule $\boldsymbol{s}$. That is a graph where each task is represented as a node. There is an arc from node x to node y, if and only if, $x = PJ_y$ or $x = PM_y$. Additionally, there are two dummy nodes, 0 and E, such that there is arc from 0 to the first task of each job, and also from the last task of each job to E. Each arc (x, y) is labelled with the processing time p_x. A critical path in $G(s)$ is the longest path from 0 to E and its length determines the makespan. All arcs that belong to a critical path are called critical arcs. In González Rodríguez et al. (2008), this idea is adapted and extended to the Fuzzy JSP by using three parallel graphs, where arcs on each one of them are labelled with each of the components of the Triangular Fuzzy Numbers (TFN). Within this neighbourhood structure, all neighbours are feasible and the connectivity property holds. For our algorithm, we take that idea and adapt it to the framework of interval uncertainty by using two parallel graphs G_1, G_2 to represent each solution. The former labels the arcs with the lower bound of the processing times and the latter with the upper bounds. Therefore, critical paths in G_1 and G_2 determine $\underline{C}_{max}$ and $\overline{C}_{max}$ respectively. We define our

neighbourhood as the set of solutions that result from reversing an arc (x, y) that is critical in G_1 or G_2 (or both).

Given that the aim is to maintain a good solution diversity in our solving method, we use a simple hill-climbing algorithm to guide the search. In this approach, neighbours of the current solution are explored in a random order until we find one that improves the current solution. That neighbour becomes the new current solution and the process is repeated until a solution with no improving neighbours is found. This method is among the fastest in the family of Local Search, since it does not necessarily evaluate all neighbours of each solution and it does not provide too much exploitation, thus helping us improve our solutions without losing much diversity.

3.4 Scouting and replacement

After each bee has found and evaluated a new food source, and the Local Search has been applied to it if the option is available, it shares the new solution with the rest of the hive. If the new food source is equivalent to the queen's one (i.e. the best food source found so far), it is discarded for the sake of maintaining diversity in the pool. Otherwise, if it improves the food source currently assigned to the bee, the bee moves to the new food source for the upcoming iteration. Similarly, if it is better than the best food source found so far, it replaces it and it is assigned to the queen. On the other hand, if it cannot improve the current food source of the bee, the number of improvement trials *fs.numTrials* of the food source is increased by one.

If the food source reaches the maximum number of improvement trials NT_{max}, it is discarded and the bee is in charge of finding a replacement. In this case, a random solution is generated following the same criteria as in Sect. 3.1 and the bee is assigned to it.

4 Experimental results

In this section, the proposed fast Elite ABC algorithm (*fEABC*) is evaluated and compared with the state-of-the-art methods. Firstly, a parametric tuning is carried out to find the best setup for the algorithm. Once found, it is compared to best known methods from the literature for the Interval JSP. Finally, a sensitivity analysis is conducted to assess the behaviour of the algorithm on instances with different amounts of uncertainty. We evaluate our method over 12 instances from the literature Díaz et al. (2020). Namely FT10 (10×10), FT20 (20×5), La21, La24, La25 (15×10), La27, La29 (20×10), La38, La40 (15×15), ABZ7, ABZ8, and ABZ9 (20×15). Values in brackets denote the instance size ($n \times m$). All experiments are done using a C++ implementation on a PC with Intel Xeon Gold 6132 processor at 2.6 Ghz and 128 Gb RAM with Linux (CentOS v6.10). For every experiment, we consider 30 runs of the method on each instance, so the resulting data are representative of the method's performance.

4.1 Parameter setup

For the parameter setup, we perform two different tuning processes depending on the use or not of Local Search. To differentiate them, we refer to the variant with Local Search as $fEABC_{LS}$, while we use simply *fEABC* for the one without the Local Search. The stopping criterion is set in both cases to $maxIter = 25$ consecutive iterations without improving the best solution found so far and the population size is set to 250 individuals according to the results obtained in Díaz et al. (2022) for ABC_{E3}. For the remaining parameters, the following values are tested:

- Decoding SGS: Semi-active, **Insertion** (see Sect. 3.1)
- Local exploration: **Insertion**, Inversion, Swap (see Sect. 3.2)
- Global attraction: **GOX**, JOX, PPX (see Sect. 3.2)
- Max. number of trials *fs.numTrials*: 10, 15, **20**
- Elite size: **40**, 50, 60

We begin the parameter tuning using a default setup with the values highlighted in bold in the list. Then we follow a sequential process where we select a parameter and test all its possible values. Once the best value for that parameter is found, it is set and the process repeats until all parameters have been established. Table 1 displays the best resulting configuration for each variant.

Regarding the use of Semi-active SGS or Insertion SGS, our results show that using an insertion strategy, and thus moving in the search space of active schedules, is better in general. In fact, the best setup for *fEABC* using the Insertion SGS obtains makespan values that are 7.2% better in average than those obtained with the best setup using the Semi-active SGS. When including Local Search, using Semi-active schedules brings more diversity, which could potentially benefit the exploitation of LS. However, this is not the case, and using the Insertion SGS still gets results that are 5.0% better than using the Semi-active SGS.

Table 1 Parameter setup for each variant of *fEABC*

Instance	fEABC	$fEABC_{LS}$
Decoding SGS	Insertion	Insertion
Local exploration	Insertion	Swap
Global attraction	JOX	JOX
Improvement trials	20	15
Elite size	40	40

Table 2 Relative error (%) w.r.t. LB obtained by 30 runs of *GA*, ABC_{E3} and *fEABC* and average runtime in seconds

Instance	LB	*GA*			ABC_{E3}			*fEABC*		
		Best	Avg.(SD)	Time	Best	Avg.(SD)	Time	Best	Avg.(SD)	Time
ABZ7	656	6.3	12.5 (2.0)	1.8	5.3	7.3 (1.1)	4.5	3.7	**6.6** (1.1)	5.9
ABZ8	645	11.3	18.5 (2.1)	1.8	9.0	12.1 (1.2)	4.2	9.1	**11.1** (1.0)	6.6
ABZ9	661	13.0	18.0 (2.4)	2.2	9.7	13.1 (1.6)	6.0	9.1	**11.6** (0.9)	6.8
FT10	930	1.8	5.2 (2.1)	0.5	1.1	4.1 (1.3)	1.6	1.0	**3.5** (1.2)	1.9
FT20	1165	1.5	4.4 (1.4)	0.7	0.7	**1.7** (0.7)	2.7	0.8	1.8 (0.6)	2.7
LA21	1046	3.2	5.0 (1.3)	1.1	2.6	5.0 (1.3)	1.8	1.7	**4.2** (0.9)	2.6
LA24	935	4.1	6.3 (1.6)	0.8	2.2	5.1 (1.3)	2.7	3.4	**4.9** (1.1)	2.6
LA25	977	1.9	5.1 (2.4)	1.0	1.9	3.9 (0.9)	2.4	1.1	**3.4** (1.3)	2.7
LA27	1235	4.6	10.2 (2.0)	1.3	2.8	4.7 (1.0)	4.1	3.0	**4.6** (1.2)	5.0
LA29	1152	11.1	14.2 (1.6)	1.1	5.5	8.6 (1.3)	4.4	4.8	**7.4** (1.3)	5.4
LA38	1196	6.0	9.2 (2.3)	1.4	4.5	6.9 (1.5)	6.0	3.0	**6.1** (1.5)	4.0
LA40	1222	5.1	8.7 (2.3)	1.2	1.9	**4.2** (1.1)	3.0	3.0	4.6 (0.9)	3.5

Bold indicates the best Average Relative Error per instance among the three comparedmethods

4.2 Comparison with state-of-the-art methods

To the best of our knowledge, the most successful algorithms in the literature for solving the Interval JSP are the genetic algorithm from Díaz et al. (2020) (*GA*), the ABC_{E3} from Díaz et al. (2022), and the more recent *ESABC* from Díaz et al. (2023).

Our first target is to assess if the new method increases the population's diversity enough to allow the algorithm to converge for more iterations and reach better areas of the search space. To do so, in Table 2 we compare the *fEABC* without the Local Search, with *GA* and the ABC_{E3} method that defines the starting point for this work. For each instance, the best-known Lower Bound (*LB*) for the expected makespan is reported Díaz et al. (2022). For each method, the table displays the Relative Error (*RE*) with respect to LB of the expected makespan of the best solution obtained in 30 runs, together with the average relative error (standard deviation in brackets) among those runs and the average runtime in seconds. Best average values are highlighted in bold. We can see that in average, *fEABC* obtains the best results in 10 out of 12 instances. Not only that, but in average, the relative errors obtained by *fEABC* are 7.2% better than those obtained with ABC_{E3}, and 39.1% better than the *GA*.

Regarding runtime, despite reducing the number of evaluations in the proposed method, it takes 16% longer in average to converge than ABC_{E3}. This is an expected result, since the target is to increase diversity for the algorithm not to get easily stuck in local optima and explore further into the search space. If we analyse the speed of the algorithms per iteration, we see that an iteration in *fEABC* is actually 9% faster than an iteration of ABC_{E3}. But *fEABC* is capable of iterating for longer before meeting the stopping criterion. This is illustrated in Fig. 2, where we can see the evolution during 200 iterations of the average expected makespan in 30 runs on instance *La*29. We can see how ABC_{E3} quickly finds good quality solutions, but then gets stuck in local optima. On the other hand, *fEABC* focuses more on exploration on the early iterations, which then allows it to converge to better solutions than ABC_{E3} in the long term.

In Díaz et al. (2023), an *ESABC* incorporates a Simulated Annealing-based strategy to improve population's diversity, obtaining the best-known results for the IJSP at the cost of increasing the algorithms runtime. In Table 3 we compare both *fEABC* and $fEABC_{LS}$ with this method. First, we observe that *fEABC* has a very similar behaviour to *ESABC* in terms of average relative errors. We conduct a statistical examination to detect if there is a significant difference between them. If the samples on each instance meet the Shapiro-Wilk test of normality, an Analysis of Variance (ANOVA) is executed, followed by Tukey's Honest Significant Difference to display the results of all pairwise comparisons within the tested groups. If the test of normality fails, a Kruskall-Wallis rank sum test is performed followed by a multiple comparison to identify which groups differ. The tests show that there is a significant difference between *ESABC* and *fEABC* in only 1 of the 12 instances. However, the runtime of *fEABC* is 13.2% shorter than *ESABC*. That is, *fEABC* obtains very similar results in less time than *ESABC*, which is a significant achievement taking into account that *ESABC* incorporates a Simulated Annealing-like technique for diversity and exploitation. In $fEABC_{LS}$ we try to invest the time reduction of *fEABC* on exploitation. Reported results show that in

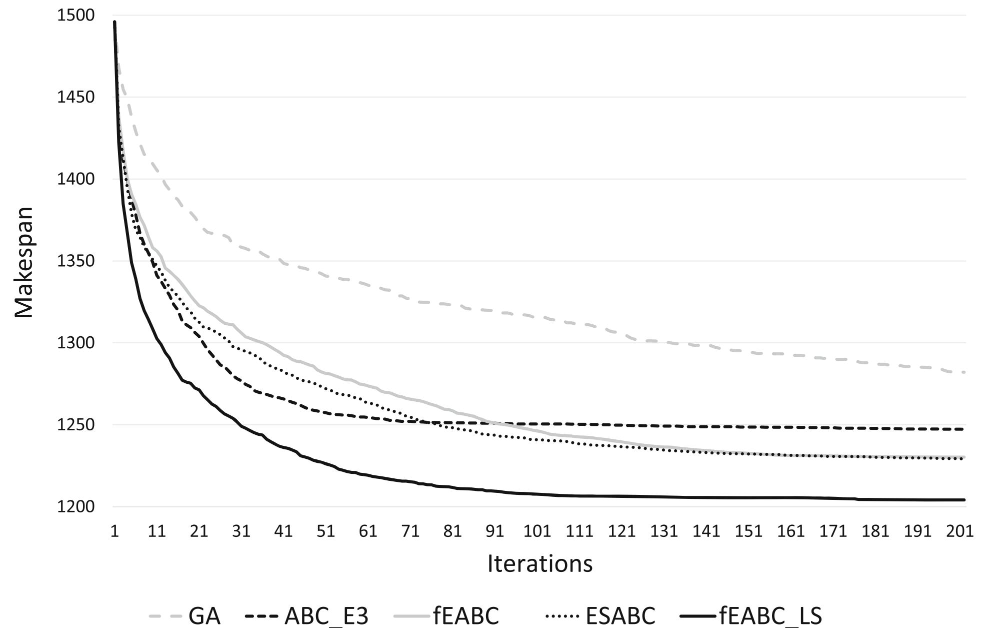


Fig. 2 Evolution of Expected makespan over 200 iterations of *GA*, ABC_{E3}, *fEABC*, *ESABC* and $fEABC_{LS}$ on instance La29

Table 3 Relative error (%) w.r.t. LB obtained by 30 runs of *ESABC*, *fEABC* and $fEABC_{LS}$ and average runtime in seconds

Instance	*ESABC*			*fEABC*			$fEABC_{LS}$		
	Best	Avg.	Time	Best	Avg.	Time	Best	Avg.	Time
ABZ7	4.2	6.7 (1.7)	6.5	3.7	6.6 (1.1)	5.9	4.1	**6.2** (1.3)	8.1
ABZ8	8.7	10.9 (1.7)	7.7	9.1	11.1 (1.0)	6.6	7.9	**10.9** (2.0)	8.2
ABZ9	8.5	**11.2** (1.5)	8.7	9.1	11.6 (0.9)	6.8	9.0	11.5 (1.3)	8.2
FT10	0.6	3.0 (1.2)	1.8	1.0	3.5 (1.2)	1.9	0.6	**2.9** (1.5)	1.8
FT20	0.7	1.8 (0.6)	2.9	0.8	1.8 (0.6)	2.7	0.7	**1.3** (0.2)	3.7
LA21	2.1	4.0 (0.7)	2.9	1.7	4.2 (0.9)	2.6	2.2	**3.7** (0.2)	3.5
LA24	3.5	5.0 (1.0)	2.6	3.4	4.9 (1.1)	2.6	3.6	**4.4** (0.7)	3.1
LA25	1.3	2.7 (0.9)	3.0	1.1	3.4 (1.3)	2.7	1.1	**2.1** (0.8)	3.0
LA27	2.8	4.1 (1.0)	5.8	3.0	4.6 (1.2)	5.0	1.9	**3.6** (1.3)	6.7
LA29	5.0	7.0 (1.1)	6.7	4.8	7.4 (1.3)	5.4	3.7	**4.8** (1.0)	7.2
LA38	3.0	5.8 (1.4)	7.2	3.0	6.1 (1.5)	4.0	3.0	**5.4** (1.1)	4.8
LA40	2.7	4.1 (0.9)	3.7	3.0	4.6 (0.9)	3.5	2.6	**4.0** (0.9)	4.0

Bold indicates the best Average Relative Error per instance among the three comparedmethods

average, $fEABC_{LS}$ obtains better results than *ESABC* in 11 out of 12 instances. Moreover, $fEABC_{LS}$ improves the RE w.r.t. LB of *ESABC* in 11.3% while the runtime increases only in 7.7%. When comparing to its counterpart without Local Search, $fEABC_{LS}$ obtains results that are 16.7% better than *fEABC* using 24% more time. These values increase to 22.8% and 45.1% respectively when compared with ABC_{E3}.

The behaviour of all different methods can be better appreciated in Fig. 2. We can observe how $fEABC_{LS}$ converges better than any of the other methods, including *ESABC*, due to its ability to balance the exploration and exploitation of the search space, being less likely to get trapped in local optima. Furthermore, *fEABC* shows a very similar behaviour to *ESABC* while being faster as shown in Table 3.

4.3 Sensitivity analysis

Finally, we carry out a sensitivity analysis to determine if taking into account uncertainty during the optimisation process is a beneficial effort in the face of increasing uncertainty. We consider a new version of $fEABC_{LS}$, $fEABC^{C}_{LS}$, where the duration of each task is taken as the midpoint of the interval. That is, uncertainty is not taken into account during the optimisation process. In this setting, makespan values obtained by $fEABC_{LS}$ are intervals,

Table 4 Average $\overline{\epsilon}$ values ($\times$1000) for $fEABC^{C}_{LS}$ and $fEABC_{LS}$ increasing processing times' interval width in +20% and +40% (standard deviation in brackets)

Instance	$fEABC^{C}_{LS}$			$fEABC_{LS}$		
	+0%	+20%	+40%	+0%	+20%	+40%
ABZ7	11.72 (1.90)	12.75 (1.60)	15.66 (1.89)	9.01 (1.23)	9.30 (1.11)	10.74 (1.04)
ABZ8	10.75 (1.84)	11.36 (1.87)	13.96 (2.15)	7.81 (1.01)	7.19 (0.90)	8.61 (1.34)
ABZ9	10.48 (1.89)	11.45 (1.65)	14.16 (1.89)	7.22 (0.82)	7.23 (0.68)	8.41 (1.04)
FT10	11.97 (1.31)	13.45 (1.75)	16.42 (2.12)	9.62 (0.98)	10.12 (1.24)	11.92 (1.77)
FT20	9.65 (1.58)	10.72 (1.64)	12.94 (1.99)	7.7 (0.42)	8.69 (0.52)	9.94 (0.55)
LA21	14.41 (1.29)	16.70 (1.51)	20.35 (1.72)	9.61 (0.96)	11.28 (1.25)	13.15 (1.31)
LA24	15.53 (2.28)	19.04 (2.04)	23.19 (2.39)	12.65 (1.85)	14.43 (2.18)	16.76 (2.23)
LA25	12.68 (2.00)	15.74 (3.00)	19.28 (3.55)	10.9 (1.13)	11.93 (1.39)	14.51 (1.38)
LA27	13.04 (1.62)	15.41 (1.80)	18.89 (2.07)	9.79 (1.16)	11.38 (0.77)	13.51 (1.48)
LA29	13.07 (1.62)	15.18 (1.70)	18.60 (1.96)	9.43 (1.07)	10.78 (1.14)	14.01 (1.32)
LA38	13.50 (1.49)	16.42 (1.70)	20.14 (1.95)	9.41 (1.34)	10.88 (1.43)	12.01 (1.78)
LA40	13.33 (1.96)	16.48 (2.76)	20.34 (3.22)	9.95 (0.90)	11.96 (1.38)	13.01 (1.20)

but makespan values obtained by $fEABC^{C}_{LS}$ will be crisp, so a straightforward comparison is not fair. Instead, we evaluate the performance of the obtained solutions in terms of their $\overline{\epsilon}$ robustness in $K = 1000$ different configurations (see Sect. 2.2). To also evaluate their robustness in more uncertain environments, we generate two new versions of each instance where the interval widths are respectively enlarged by 20% and 40%. The changes are applied symmetrically on both sides of the intervals to maintain the significance of the midpoint. If the increase on the width of an interval $\mathbf{p_o}$ would result on a negative bound, then the interval $[0, \overline{p}_o + \underline{p}_o]$ is taken instead.

Table 4 shows the $\overline{\epsilon}$ values of the solutions obtained by $fEABC_{LS}$, considering interval processing times, and $fEABC^{C}_{LS}$, considering only the midpoint of the intervals, over the three sets of instances: the original ones (+0% in the table), and the two new versions (+20% and +40% in the table). The results show that $fEABC_{LS}$ finds the most robust solutions, even when the size of the intervals is expanded by 20 and 40%. In fact, the solutions obtained by $fEABC_{LS}$ have better robustness values over the scenarios with an increase of 20% than those obtained by $fEABC^{C}_{LS}$ on the original instances. As expected, in both cases the robustness deteriorates as uncertainty increases, but there is a clear difference between incorporating uncertainty in the optimisation or not. For instance, when the intervals are increased by 20 and 40% the $\overline{\epsilon}$ values of solutions obtained with $fEABC_{LS}$ get 10.68% and 29.62% worse, whereas for $fEABC^{C}_{LS}$ the $\overline{\epsilon}$ values become 16.37% and 42.50% worse respectively.

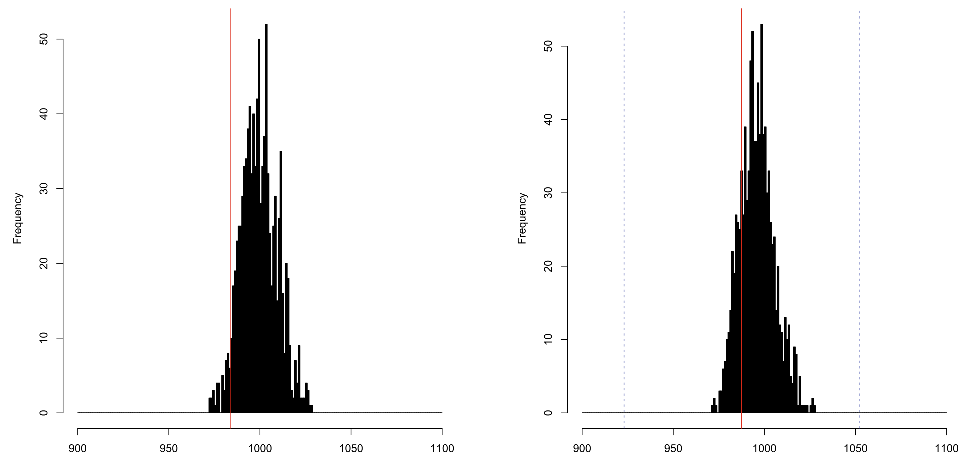
This is better illustrated in Fig. 3. Each graphic contains a histogram with the $K = 1000$ realisations of the best solution obtained in $fEABC_{LS}$ and the best one from $fEABC^{C}_{LS}$ on the different variants of instance $La25$. Red lines depict the predictive values: C_{max} when using $fEABC^{C}_{LS}$ and $E[\mathbf{C_{max}}]$ when using $fEABC_{LS}$. In the latter case, blue dotted lines show the interval makespan bounds. If we compare the graphics with the original $La25$ instance, we can see how the red line in $fEABC_{LS}$ is quite inside the histogram, while in the case of $fEABC^{C}_{LS}$ it is more on the left side, showing that the solution in this case is quite optimistic and real executions tend to have a higher makespan. When uncertainty increases, the histograms in both cases tend to spread towards the right side of the plot. However, with the $fEABC_{LS}$ solution, the red line is still quite inside the histogram, showing that it is a better predictor than $fEABC^{C}_{LS}$ where it remains on the left. Moreover, in the case of $fEABC^{C}_{LS}$, real executions tend to move more to the right side of the graphic than with $fEABC_{LS}$, starting to accumulate around 1050 in $La25_{+40\%}$.

5 Conclusions

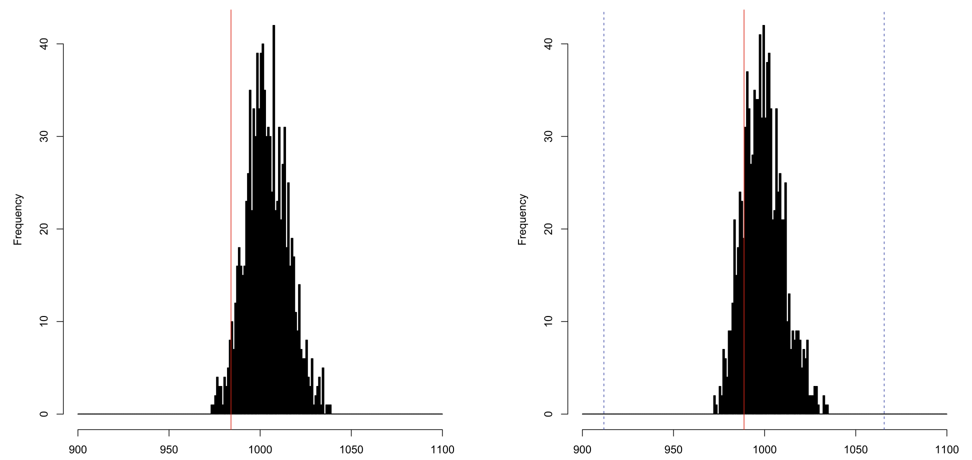
We have considered the IJSP, a version of the JSP that models the uncertainty on task durations appearing in real-world problems using intervals. In Díaz et al. (2022) we proposed an ABC algorithm tailored to this problem. In that study, diversity issues where spotted and a new selection mechanism *Elite*3 was proposed to tackle them. In this work, we extend the mentioned *ABC* by including new diversity strategies and modifying the general structure of the algorithm to reduce the number of unnecessary evaluations. Exploration is more encouraged before reaching the evaluation and replacement phases. At the same time, the number of evaluation phases is reduced from three to one. Moreover, the number of parameters to set up in the algorithm is greatly reduced, making it easier to tune for different environments.

A parametric analysis showed that using semi-active schedules brings in general more diversity to the

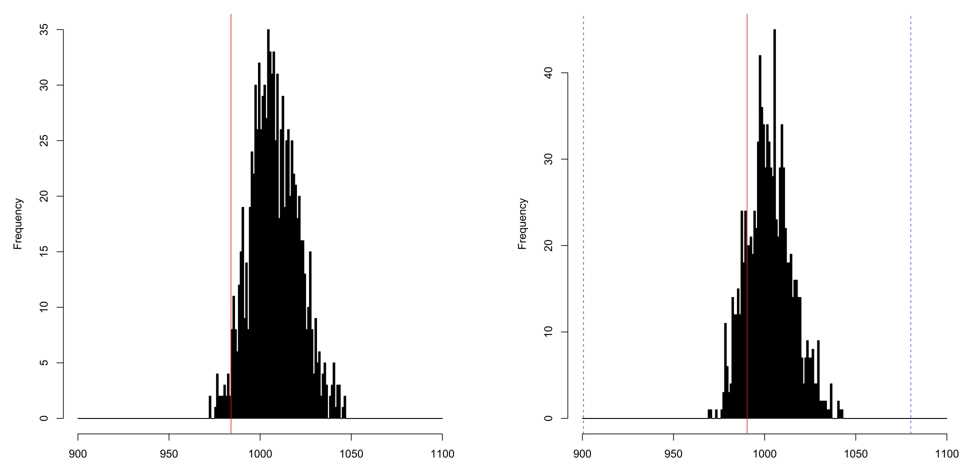
Fig. 3 Histograms of C_{max}^{ex} obtained with the best solutions from $fEABC_{LS}^{C}$ and $fEABC_{LS}$ on $K = 1000$ configurations of instances $La25$, $La25_{+20\%}$ and $La25_{+40\%}$



(a) $La25$ ($fEABC_{LS}^{C}$) (b) $La25$ ($fEABC_{LS}$)



(c) $La25_{+20\%}$ ($fEABC_{LS}^{C}$) (d) $La25_{+20\%}$ ($fEABC_{LS}$)



(e) $La25_{+40\%}$ ($fEABC_{LS}^{C}$) (f) $La25_{+40\%}$ ($fEABC_{LS}$)

population, but it lacks enough exploitation, thus using an insertion SGS capable of generating active schedules provides better results overall. The proposed solving method was favourably compared with its previous version ABC_{E3} and obtained similar results to the best method in the IJSP literature while using significantly less time. The reduction in runtime and the increase of diversity allowed us to hybridize our method with a Hill Climbing algorithm. As expected, the runtime increases, but the improvement in solution quality is larger than the time increase and leads the algorithm to the best results for the IJSP, outperforming all previously published methods.

A sensitivity analysis was also performed to assess the robustness of the obtained solutions in environments with larger amounts of uncertainty. The comparison was also made to see the advantages of considering the uncertainty during the optimisation process. The results showed that in that case, the robustness of the obtained solutions is much better than solutions obtained when solving the problem without taking the uncertainty into account.

Acknowledgements This research has been supported by the Spanish Government under research Grant PID2019-106263RB-I00 and by the Asturian Government under research grant Severo Ochoa.

Author contributions Authors are listed following the signing order. Conceptualization: HD, JJP, IG-R, Methodology: HD, JJP, IG-R; Formal analysis and investigation: HD, JJP, IG-R; Writing: HD, JJP, CRV; Funding acquisition: IG-R, CRV; Resources: IG-R, CRV; Supervision: JJP, IG-R.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability All instances used in this study are taken from the literature, and therefore available to the scientific community. Detailed data on obtained results is available upon reasonable request to the corresponding author.

Declarations

Funding This research has been supported by the Spanish Government under research grant PID2019-106263RB-I00 and by the Asturian Government under research grant Severo Ochoa.

Conflict of interest The authors declare that they have no financial or non-financial conflicts of interest.

Ethics approval Not applicable

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Allahverdi A, Aydilek H, Aydilek A (2014) Single machine scheduling problem with interval processing times to minimize mean weighted completion time. Comput Oper Res 51:200–207. https://doi.org/10.1016/j.cor.2014.06.003

Banharnsakun A, Sirinaovakul B, Achalakul T (2012) Job shop scheduling with the best-so-far ABC. Eng Appl Artif Intell 25(3):583–593. https://doi.org/10.1016/j.engappai.2011.08.003

Bidot J, Vidal T, Laboire P (2009) A theoretic and practical framework for scheduling in stochastic environment. J Sched 12:315–344

Bierwirth C, Mattfeld DC, Kopfer H (1996) On permutation representations for scheduling problems. In: PPSN IV: Proceedings of the 4th international conference on parallel problem solving from nature, pp 310–318. Springer London, UK

Bierwirth C (1995) A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum 17:87–92

Díaz H, González-Rodríguez I, Palacios JJ, Díaz I, Vela CR (2020) A genetic approach to the job shop scheduling problem with interval uncertainty. In: Lesot M-J, Vieira S., Reformat MZ, Carvalho JP, Wilbik A, Bouchon-Meunier B, Yager RR (eds.) Information processing and management of uncertainty in knowledge-based systems, pp 663–676. Springer. https://doi.org/10.1007/978-3-030-50143-3_52

Díaz H, Palacios JJ, González-Rodríguez I, Vela CR (2023) An elitist seasonal artificial bee colony algorithm for the interval job shop. Integrated Comput-Aid Eng 1–20. https://doi.org/10.3233/ICA-230705

Díaz H, Palacios JJ, González-Rodríguez I, Vela CR (2022) Elite artificial bee colony for makespan optimisation in job shop with interval uncertainty. In: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., Paz López, F., Adeli, H. (eds.) Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence, pp. 98–108. Springer. https://doi.org/10.1007/978-3-031-06527-9_10

Díaz H, Palacios JJ, Díaz I, Vela CR, González-Rodríguez I (2022) Robust schedules for tardiness optimization in job shop with interval uncertainty. Logic J IGPL. https://doi.org/10.1093/jigpal/jzac016

Fortemps P, Roubens M (1996) Ranking and defuzzification methods based on area compensation. Fuzzy Sets Syst 82:319–330

García-Álvarez J, González MA, Vela CR, Varela R (2018) Electric vehicle charging scheduling by an enhanced artificial bee colony algorithm. Energies 11(10):2572. https://doi.org/10.3390/en11102752

González Rodríguez I, Vela CR, Puente J, Varela R (2008) A new local search for the job shop problem with uncertain durations. In: Proceedings of the eighteenth international conference on automated planning and scheduling (ICAPS-2008), pp 124–131. AAAI Press Sidney (2008)

Karaboga D (2005) An idea based on honey bee swarm for numerical optimization, technical report - tr06. Technical Report, Erciyes University

Karaboga D, Akay B (2009) A comparative study of artificial bee colony algorithm. Appl Math Comput 214(1):108–132. https://doi.org/10.1016/j.amc.2009.03.090

Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (ABC) algorithm

and applications. Artif Intell Rev 42:21–57. https://doi.org/10.1007/s10462-012-9328-0
Kennedy J, Eberhart R (1995) Particle swarm optimization. In: IEEE international conference on neural networks, pp 1942–1948. IEEE Press. New Jersey
Lei D (2011) Population-based neighborhood search for job shop scheduling with interval processing time. Comput Ind Eng 61:1200–1208. https://doi.org/10.1016/j.cie.2011.07.010
Lei D (2012) Interval job shop scheduling problems. Int J Adv Manuf Technol 60:291–301. https://doi.org/10.1007/s00170-011-3600-3
Li X, Gao L, Wang W, Wang C, Wen L (2019) Particle swarm optimization hybridized with genetic algorithm for uncertain integrated process planning and scheduling with interval processing time. Comput Ind Eng 235:1036–1046
Ono I, Yamamura M, Kobayashi S (1996) A genetic algorithm for job-shop scheduling problems using job-based order crossover. In: Proceedings of IEEE international conference on evolutionary computation, pp 547–552. IEEE
Palacios JJ, Vela CR, González-Rodríguez I, Puente J (2014) Schedule generation schemes for job shop problems with fuzziness. In: Schaub T, Friedrich G, O'Sullivan B (eds.) Proceedings of ECAI 2014. Frontiers in Artificial Intelligence and Applications, vol. 263, pp 687–692. IOS Press. https://doi.org/10.3233/978-1-61499-419-0-687
Pinedo ML (2016) Scheduling. Theory, algorithms, and systems, 5th edn. Springer, Cham
Sprecher A, Kolisch R, Drexl A (1995) Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. Eur J Oper Res 80:94–102
Van Laarhoven P, Aarts E, Lenstra K (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125
Wong L-P, Puan CY, Low MYH, Chong CS (2008) Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems. In: 2008 Winter simulation conference, pp 2050–2058. https://doi.org/10.4028/www.scientific.net/amm.26-28.657
Yao B, Yang C, Hu J, Yin G, Yu B (2010) An improved artificial bee colony algorithm for job shop problem. Appl Mech Mater 26–28:657–660. https://doi.org/10.4028/www.scientific.net/amm.26-28.657

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.